charles river analytics

Charles River Analytics Inc.
625 Mount Auburn St.
Cambridge, MA 02138

www.cra.com

p: 617.491.3474
f: 617.868.0780

15 MAR 2011

Reference: Government Contract No. N00014-09-C-0050, "Enhancing Simulation-based Training Adversary Tactics via Evolution (ESTATE)"
Charles River Analytics Contract No. C08098

Subject: Contractor's Status Report: Quarterly Status Report #9
Reporting Dates: 12/15/2010 – 3/15/2011

Dear Dr. Hawkins,

The following is the Contractor's Quarterly Status Report for the subject contract for the indicated period. During this reporting period we have concentrated on Task 4: Develop Trainee Model Processing, Task 6: Simulation-based Training System Integration and Task 8: Transition

1. Summary of Progress

1.1 Transition Opportunities with USMC TECOM

During the indicated period, we have been in discussions with USMC Training and Education Command (TECOM) MAGTF Training Simulations Division (TSD) to inform the community of the training systems capabilities being developed under ESTATE along with the PROMPTER framework. Our discussions have led to potential interest from several different avenues including Marine Corps University, College of Distance Education and Training (CDET), the Marine Corps Warfighting Laboratory (MCWL), USMC Center for Advanced Operational Culture Learning (CAOCL), and PM Training Systems (TRASYS). To better describe how the pieces fit together, a high-level diagram was provided along with overview materials and movies for each capability. This diagram is displayed below in Figure 1. The architecture is divided into two areas. The left-hand side describes the layered model for providing microgame-based training while the right-hand side describes the layered model for providing simulation-based training. Each training method is divided into a content, execution, and delivery layer. At the content layer, user communities (e.g., training staff) develop the training material. At the execution layer, the system provides the necessary logic that backs the training experience. The trainee participates in the training via the delivery layer. For microgame-based training, this may be delivered via a mobile, web-based, or desktop application. For simulation-based training, this is provided through a simulation environment (e.g., VBS2). ESTATE currently sits as an additional service in the execution layer. The ESTATE Adaptation Engine modifies the content to maximize training efficacy. Our current work is integrating with the microgame-based training system on the left side, using the performance history of the trainee interacting with the PROMPTER framework to adapt challenges. Future integration could also support the ESTATE

Report Documentation Page

Form Approved
OMB No. 0704-0188

1. REPORT DATE	2. REPORT TYPE	3. DATES COVERED
15 MAR 2011		**00-00-2011 to 00-00-2011**

4. TITLE AND SUBTITLE	5a. CONTRACT NUMBER
Enhancing Simulation-based Training Adversary Tactics Via Evolution (ESTATE)	5b. GRANT NUMBER
	5c. PROGRAM ELEMENT NUMBER
6. AUTHOR(S)	5d. PROJECT NUMBER
	5e. TASK NUMBER
	5f. WORK UNIT NUMBER
7. PERFORMING ORGANIZATION NAME(S) AND ADDRESS(ES) **Charles River Analytics, Inc,625 Mount Auburn St.,Cambridge,MA,02136**	8. PERFORMING ORGANIZATION REPORT NUMBER
9. SPONSORING/MONITORING AGENCY NAME(S) AND ADDRESS(ES)	10. SPONSOR/MONITOR'S ACRONYM(S)
	11. SPONSOR/MONITOR'S REPORT NUMBER(S)

12. DISTRIBUTION/AVAILABILITY STATEMENT
Approved for public release; distribution unlimited

13. SUPPLEMENTARY NOTES

14. ABSTRACT

15. SUBJECT TERMS

16. SECURITY CLASSIFICATION OF:			17. LIMITATION OF ABSTRACT	18. NUMBER OF PAGES	19a. NAME OF RESPONSIBLE PERSON
a. REPORT	b. ABSTRACT	c. THIS PAGE			
unclassified	**unclassified**	**unclassified**	**Same as Report (SAR)**	**7**	

adaptation engine with the Persona™ run-time engine for providing sophisticated, intelligent behavior to virtual characters in simulated environments.

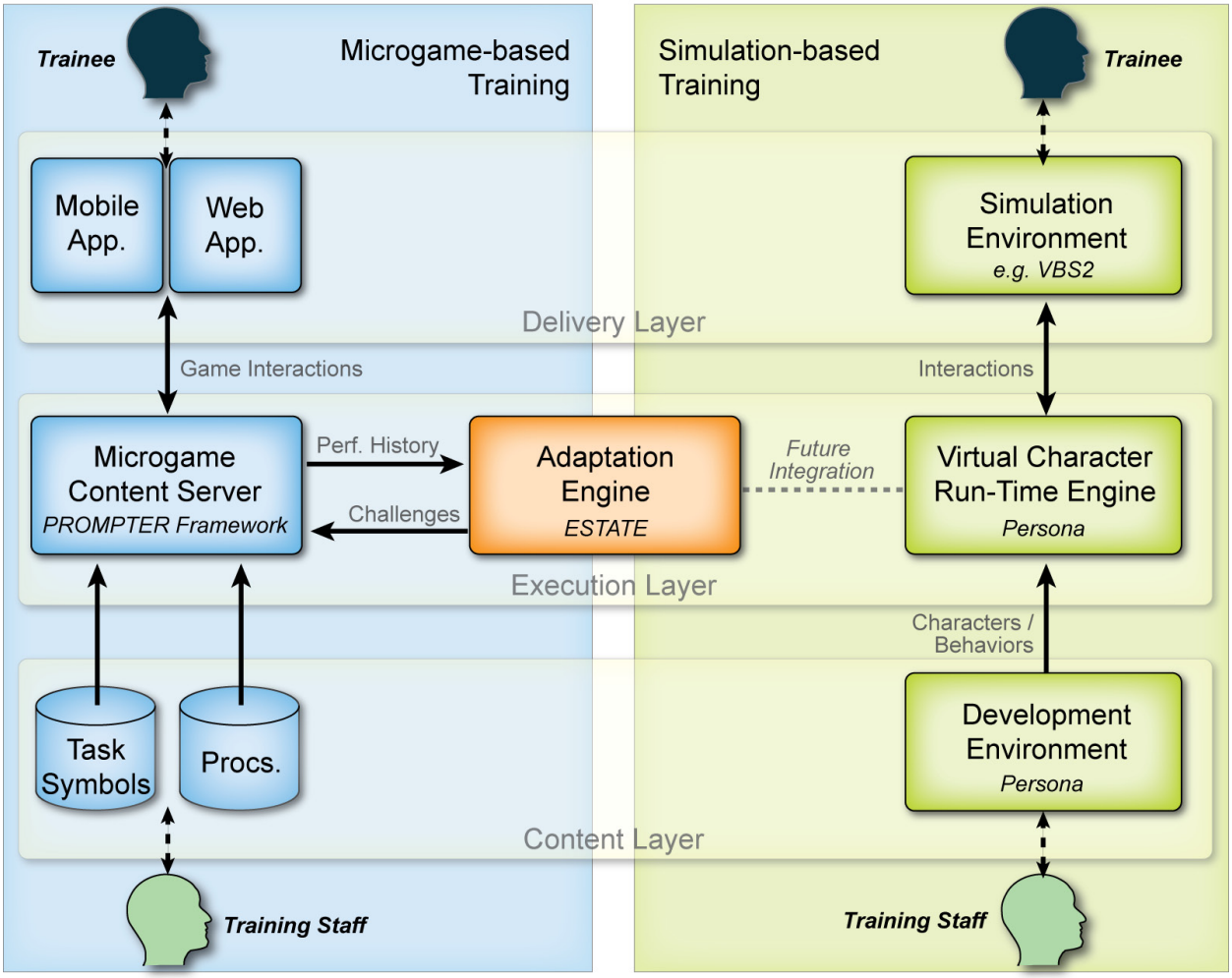

Figure 1: Integrated Training System Architecture

1.2 Integrating ESTATE with PROMPTER Framework

Last reporting period, we elected to integrate ESTATE with the PROMPTER framework, an on-going effort funded by the U.S. Army Aeromedical Research Laboratory (USAARL) that uses a microgame-based training to improve the comprehension and recall of first-aid procedures. PROMPTER provides ESTATE with a well-defined challenge domain to apply adaptive training. However, the PROMPTER framework needed to be expanded to support adaptive training. The envisioned integration design is shown in Figure 2.

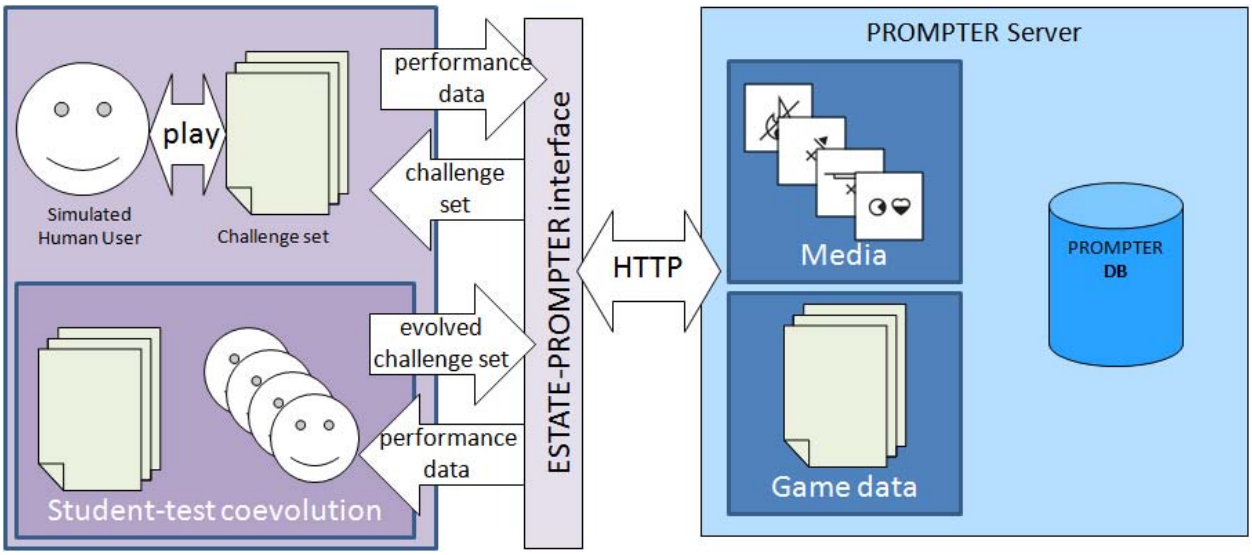

Figure 2. ESTATE-PROMPTER simulation framework integration design

As described in the last progress report, the PROMPTER server is a RESTful database used to store progress made by players as they progress through the game challenges. Originally, the PROMPTER framework only supported the ability to store progress, there was no method by which to specify the challenges that players were given. During the indicated period, the PROMPTER server has been expanded to include challenge sets. Each challenge set is a collection of challenges created by ESTATE. When players (real or simulated) play the game, the game client fetches the latest challenge set from the server and presents the challenges to the player. The integration with the server is now complete, we can now store and retrieve challenges on the PROMPTER server as well as the underlying game data that makes up each challenge.

The PROMPTER server currently stores game data as a list of mnemonics. Each mnemonic contains a symbol, its meaning, and a list of distractor symbols and meanings (i.e. symbols that are visually similar to the mnemonic's symbol and meanings that are conceptually similar to the symbol's meaning.) Using this list of mnemonics, we can create game challenges that are made up of a cue and four possible responses, one of which is correct.

1.3 Player Modeling and Challenge Generation Implementation

Now that we have updated the PROMPTER server to support third-party challenge specification, our next task was to implement a testing framework to support simulated players, challenges, and performance.

To meet these needs, we must implement (1) simulated players that can play challenges and (2) a simulated PROMPTER client that can retrieve challenges from the server and post the simulated player's actions. We will also be testing multiple challenge generation algorithms. In order to expedite this process we have created a uniform interface for simulated players and challenge generators to easily swap in and out different models and algorithms. These interfaces are designed to be general enough to cover a wide range of implementations while still remaining

fairly complete, so that each implementation acts as a 'black box' that can simply be fed information and results can be returned. Each simulated player can load and save its internal model for persistence, play a given challenge, and interpret feedback.

The simulated client, used by all simulated players, is a minimal implementation devoid of any graphical elements. The client simply connects to the PROMPTER server, retrieves the latest challenge sets, and presents the challenges to the player. The player makes a choice for each challenge within the challenge set and receives feedback (i.e. the player is told which choice was correct.) The simulated client then collects all the player data and posts it to the PROMPTER client as a session and a collection of logs.

1.3.1 Player Models

With our framework in place, our next task was to develop a set of player models to be used for testing. We developed two player model implementations, a simple player and advanced player.

Our first player model, dubbed Simple Player, is used primarily for code testing purposes. This player randomly guesses when it does not know the answer, and permanently learns the correct answer when it is given feedback. Since this player does not model human learning, its purpose is only to verify and validate the ESTATE implementation behaves as expected.

Our second player model is a more advanced player. This player uses an association matrix to make choices. The association matrix consists of a column for each symbol and a row for each meaning. Each value [*m*,*n*] in the matrix represents the player's association that symbol *m* is the correct answer for meaning *n*. When presented with a challenge, which consists of a question and four possible answers, the player will find the values for each combination and choose the symbol with the highest score. A number of learning algorithms can be implemented to adjust the matrix's values when feedback is given. The initial learning algorithm implemented provides a simple increase to the correct symbol-meaning association and a decrease to the incorrect symbol-meaning associations.

1.3.2 Challenge Generation

Now that our Player Models were created, our next task was to develop different challenge generation algorithms to compare against our coevolutionary approach. As an initial step, we formulated a general challenge generator interface that each algorithm would provide behavior for.

The challenge generator interface consists of methods for providing game data from which to generate challenges, as well as player performance history. Depending on the challenge generation algorithm, not all of this data may be used. In addition, methods are provides to begin the challenge generation process and retrieves the generated challenges. Several challenge generators have been implemented:

Random Generator - This generator generates *N* random challenges for each symbol in the list of mnemonics. Player and challenge history are not used.

Exhaustive Generator - This generator creates all possible combination of questions and symbols. Player and challenge history are not used.

Spaced Repetition Generator - Player history is reviewed so that challenges not seen by the player and challenges the player has previously failed are presented to the user before challenges the player has successfully answered in the past. Both challenge segments are randomized before they are combined into a single set of challenges. The Spaced Repetition Generator determines the order in which challenges are presented where individual challenges can be created randomly or according to a specific scheme.

During the indicated period, we also developed a challenge generation server, which will be used for all challenge generation methods. The challenge generation server is responsible for periodically retrieving player history and generating challenges. Once generated, the challenges are posted to the PROMPTER server's challenge list. For testing purposes this can be run on-demand, but there is also support for scheduling challenge generation. When scheduled, challenge generation will occur on a separate thread than the main server loop, so that challenges can be generated for multiple players. (This also prevents the server from 'freezing' when more time-intensive challenge generation algorithms are run.) The server will eventually be designed to run as a service, but currently runs as a stand-alone application.

1.3.3 Student-Test Coevolution Design

During the indicated period, we designed the ability to use student-test coevolution to construct challenge sets. This will be compared to the alternative methods previously described above. ESTATE's coevolutionary challenge generation produces an optimized challenge set for a particular trainee, given that trainee's past performance. Figure 3 shows the design of this process. First the *Performance Data and Play History* of a particular trainee is retrieved from the server data store via the **ESTATE-PROMPTER Interface to Server**. The **Trainee Modeler** uses this information to construct a model of the Trainee, including the trainee's skills and deficiencies, as well as what is unknown about potential trainee performance. The **Trainee Population Generator** uses this *Trainee Model* to create an *Initial Coevolution Population* that represents a sampling from the space of possible trainee skills. This initial population seeds the **Student-Test Coevolution** that coevolves both the possible trainee strategies and the challenges on which those strategies are tested. Because student-test coevolution searches the space of challenges incrementally by iteratively testing thousands of small changes, each single generation is only slightly improved over the previous, if at all. This process increases the difficulty of the collection of challenges, monotonically, but the process must know when this difficulty has reached the maximum to which the trainee can adjust, the Zone of Proximal Development (ZPD). Therefore, at each generation, the **ZPD Estimator** examines the *Coevolution State* to determine if the new challenges have reached sufficient difficulty for the trainee to attempt. If so, the **Challenge Set Extractor** extracts the *New Challenge Set* from the complete trace of coevolution, which may include many more iterations of challenges than can be contained in one challenge set. The resultant challenge set is transmitted to the PROMPTER server via the ESTATE-PROMPTER interface for the next play session.

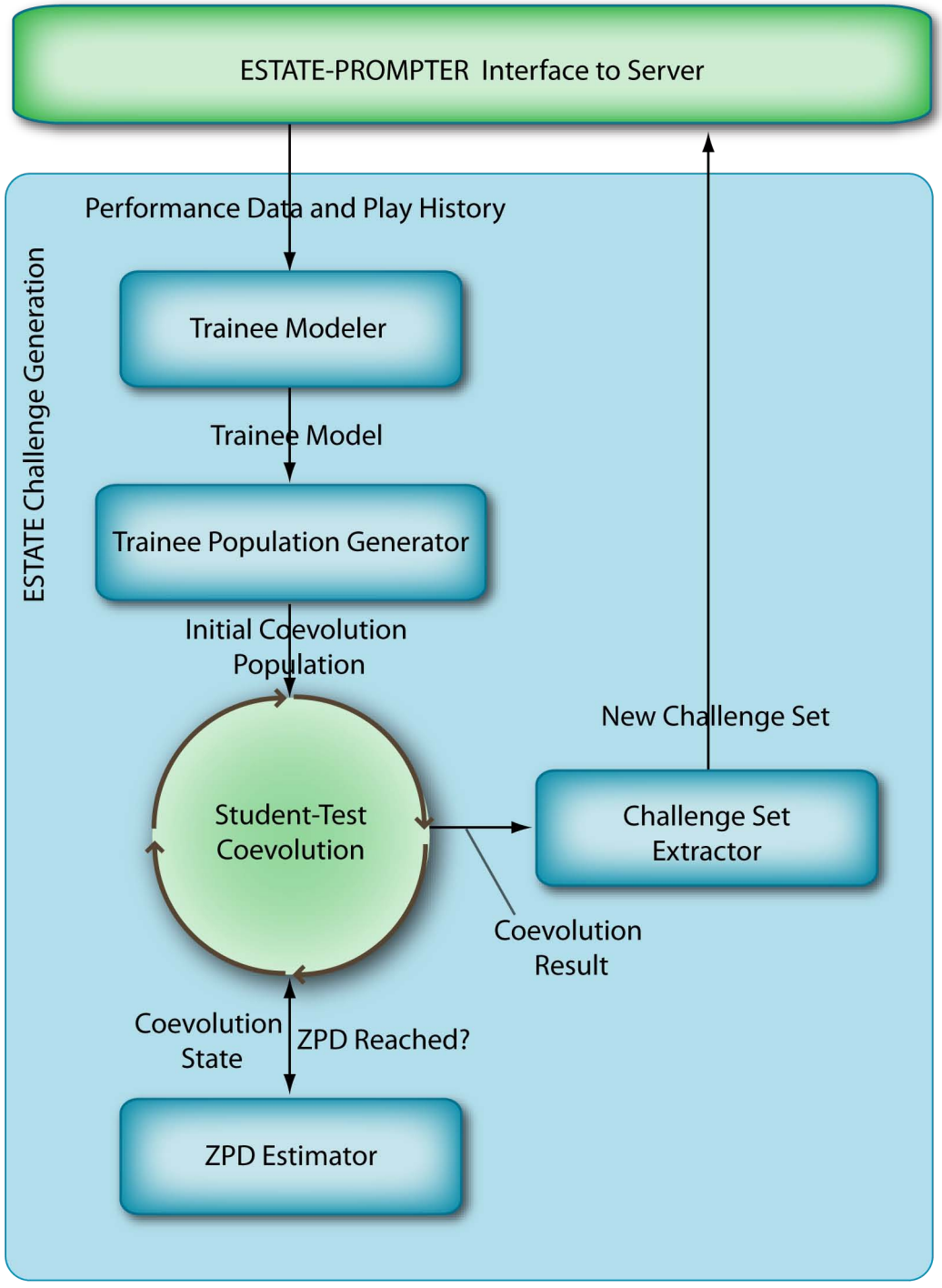

Figure 3: ESTATE challenge generation design

2. Scheduled Items

In the next reporting period we plan to address the following items:

- Complete the ESTATE Coevolutionary Challenge Generation
- Examine the performance of ESTATE algorithms within the PROMPTER framework using simulated experiments
- Develop a PROMPTER client for live participants using the ESTATE algorithms
- Continued pursuit of development and transition opportunities for the USMC Training Simulations Division

Sincerely,

Brad Rosenberg
Principal Investigator